Ex: 11 — Variational Autoencoder (VAE)

Aim: to build a variational Auto encoder that can learn compressed representations of data and generate new similar samples

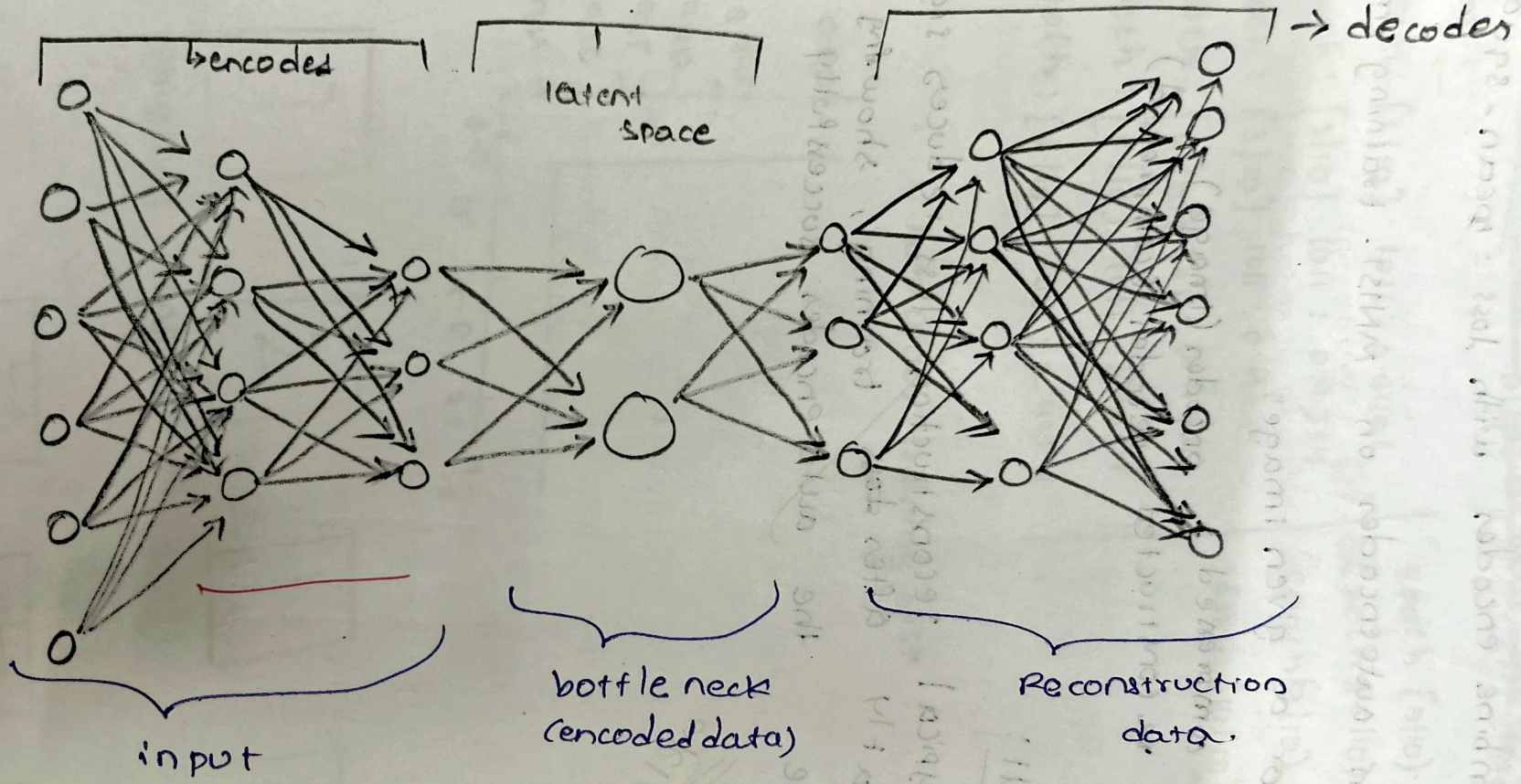
Algorithm:

1. load and normalize the dataset
2. pass data through an encoder to get two outputs
3. use the reparameterization trick.
4. pass z through a decoder to reconstruct the input
5. compute total loss = reconstruct loss + KL divergence
6. update network weights using a optimizer
7. Repeat for multiple epochs.

Pseudo code:

```
initialize encoder and decoder
for each epoch:
    for each batch of data
        mu, log_var = encoder(x)
        z = mu + exp(0.5 * log_var) * random_noise()
        x_hat = decoder(z)
        loss = reconstruction_loss(x, x_hat)
               + KL_divergence(mu, log_var)
```

Variational auto encoder architecture.



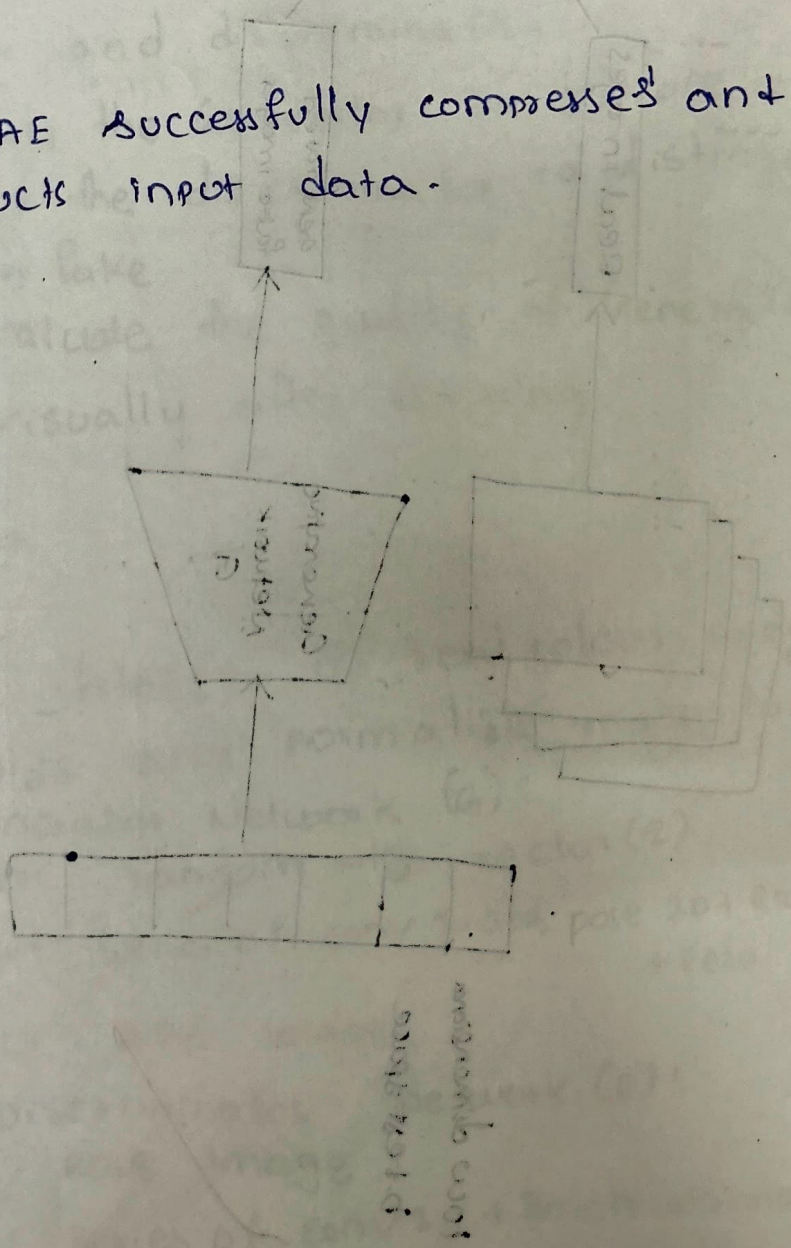
update.weights(loss)

observation:

* the loss decreases with training.
* reconstructed images become clearer after several epochs
* model learns a smooth latent space. similar datapoint cluster together

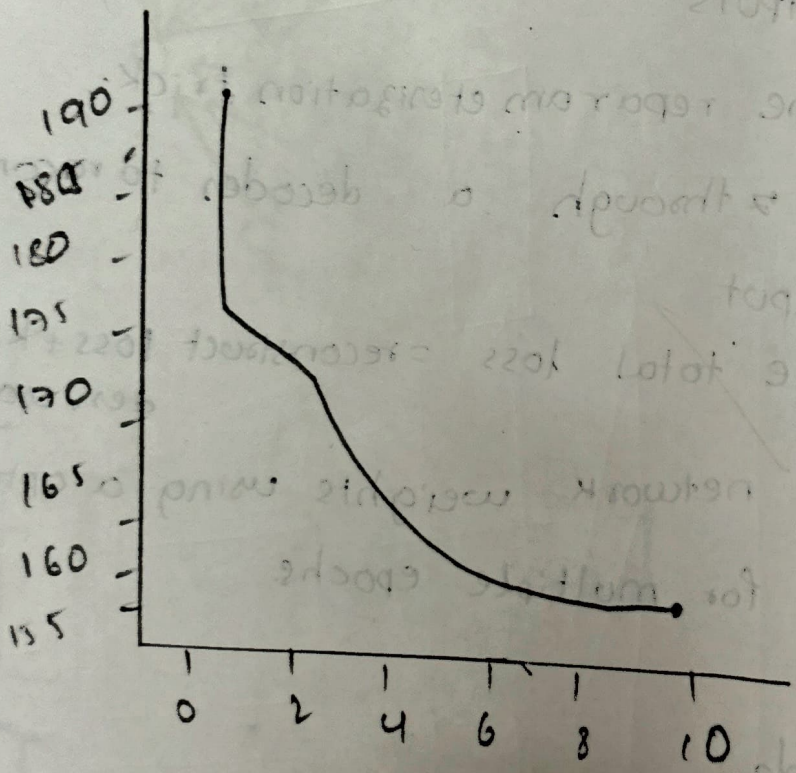
Result:

* the VAE successfully compresses and reconstructs input data.



Epoch [1/10] loss 194.10
Epoch [2/10] loss 168.25
Epoch [3/10] loss 164.28
Epoch [4/10] loss 162.21
Epoch [5/10] loss 160.63
Epoch [6/10] loss :
Epoch [7/10] loss :
Epoch [8/10] loss :
Epoch [9/10] loss :
Epoch [10/10] loss :

VAE training loss



```
from tensorflow.keras.layers import Input, Dense
import matplotlib.pyplot as plt
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()       # labels not needed
x_train = x_train.astype("float32") / 255.0
x_test  = x_test.astype("float32") / 255.0

x_train = x_train.reshape(len(x_train), 784)
x_test  = x_test.reshape(len(x_test), 784)

input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)      # compressed layer

decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)  # reconstruct pixels

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mse', metrics=['accuracy'])

history = autoencoder.fit(
    x_train, x_train,
    epochs=10,
    batch_size=256,
    shuffle=True,
    validation_data=(x_test, x_test)
)

plt.figure(figsize=(6,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("Autoencoder Training Loss")
plt.xlabel("Epochs")
plt.ylabel("MSE Loss")
plt.legend()
plt.show()
decoded_imgs = autoencoder.predict(x_test)
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    plt.axis("off")

    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.title("Reconstructed")
    plt.axis("off")
plt.show()
```

Model: "encoder"

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 784)	0	-
dense_47 (Dense)	(None, 512)	401,920	encoder_input[0]…
z_mean (Dense)	(None, 2)	1,026	dense_47[0][0]
z_log_var (Dense)	(None, 2)	1,026	dense_47[0][0]
z (Lambda)	(None, 2)	0	z_mean[0][0], z_log_var[0][0]

Total params: 403,972 (1.54 MB)
Trainable params: 403,972 (1.54 MB)
Non-trainable params: 0 (0.00 B)

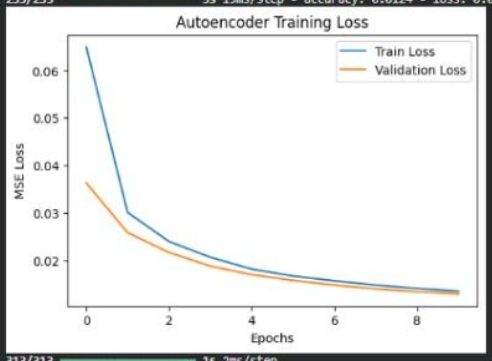
Model: "decoder"

Layer (type)	Output Shape	Param #
z_sampling (InputLayer)	(None, 2)	0
dense_48 (Dense)	(None, 512)	1,536
dense_49 (Dense)	(None, 784)	402,192

Total params: 403,728 (1.54 MB)
Trainable params: 403,728 (1.54 MB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 14s 50ms/step - loss: 210.1661 - val_loss: 0.0000e+00
Epoch 2/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 37ms/step - loss: 171.5112 - val_loss: 0.0000e+00
Epoch 3/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 42ms/step - loss: 164.4603 - val_loss: 0.0000e+00
Epoch 4/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 42ms/step - loss: 161.6138 - val_loss: 0.0000e+00
Epoch 5/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 40ms/step - loss: 159.8511 - val_loss: 0.0000e+00
Epoch 6/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 39ms/step - loss: 158.4258 - val_loss: 0.0000e+00
Epoch 7/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 11s 44ms/step - loss: 157.1902 - val_loss: 0.0000e+00
Epoch 8/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 43ms/step - loss: 156.0260 - val_loss: 0.0000e+00
Epoch 9/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 40ms/step - loss: 154.9950 - val_loss: 0.0000e+00
Epoch 10/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 39ms/step - loss: 154.0985 - val_loss: 0.0000e+00
```

```
Epoch 1/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 31ms/step - accuracy: 0.0100 - loss: 0.0978 - val_accuracy: 0.0084 - val_loss: 0.0363
Epoch 2/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - accuracy: 0.0087 - loss: 0.0329 - val_accuracy: 0.0080 - val_loss: 0.0258
Epoch 3/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - accuracy: 0.0087 - loss: 0.0249 - val_accuracy: 0.0108 - val_loss: 0.0216
Epoch 4/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - accuracy: 0.0092 - loss: 0.0213 - val_accuracy: 0.0118 - val_loss: 0.0188
Epoch 5/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.0098 - loss: 0.0186 - val_accuracy: 0.0106 - val_loss: 0.0170
Epoch 6/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - accuracy: 0.0090 - loss: 0.0170 - val_accuracy: 0.0106 - val_loss: 0.0157
Epoch 7/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - accuracy: 0.0105 - loss: 0.0159 - val_accuracy: 0.0113 - val_loss: 0.0148
Epoch 8/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.0105 - loss: 0.0149 - val_accuracy: 0.0113 - val_loss: 0.0140
Epoch 9/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - accuracy: 0.0121 - loss: 0.0141 - val_accuracy: 0.0127 - val_loss: 0.0134
Epoch 10/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - accuracy: 0.0124 - loss: 0.0136 - val_accuracy: 0.0131 - val_loss: 0.0129
```




```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```